情報処理学会論文誌　コンシューマ・デバイス & システム　Vol.8　No.2　24–33 (May 2018)

コンシューマ・デバイス論文

アプリケーションの不具合解析を効率化する走行制御向けECUプラットフォームの検討

堀田 勇樹[1,a)]　宇治土公 雄介[1]　福島 悠史[2]　成沢 文雄[2]　林 正人[2]

受付日 2017年9月30日, 採録日 2018年2月14日

概要：自動運転をはじめとした走行制御システムでは，実環境走行におけるアプリケーションの動作検証が不可欠である．動作検証中に見つかったアプリケーションの不具合を確実に再現して効率的に解析するためには，アプリケーションに対するデータ入力の再現が必要である．しかし，車両に搭載するECUでは，アプリケーションに対するデータ入力を統一的に行う基盤や，そのデータ入力を高精度に再現する方式を持たないため，走行制御システムの高度化によるアプリケーション数やセンサ数の増加にともない，アプリケーションの不具合の再現が難しいケースが増えてきている．そこで本研究では，1) リアルタイムデータベース（RTDB）を介してアプリケーションに対するデータ受渡しを統一的に行うプラットフォームと，2) ログ記録時のアプリケーション実行を基準としたデータ入力タイミングをログ再生時に再現するログ記録・再生制御方式を提案する．提案するログ記録・再生制御方式の評価を行い，従来方式と比べてアプリケーションの不具合再現にかかる時間を約60%削減できる見込みを得た．

キーワード：自動運転，ECU，開発プラットフォーム

ECU Platform Enabling Efficient Application Debugging for Driving Control System

Yuki Horita[1,a)]　Yusuke Ujitoko[1]　Yuji Fukushima[2]　Fumio Narisawa[2]　Masato Hayashi[2]

Received: September 30, 2017, Accepted: February 14, 2018

Abstract: In the process of developing the driving control system including automated driving, it is indispensable to verify the behavior of the application in actual environment. In order to precisely reproduce the defects of the application found during actual tests and efficiently analyze them, it is necessary to reproduce the data input into the application. However, the existing ECU has neither platform enabling uniform way of data input to the application, nor a method that can reproduce the data input with high precision. This makes the number of cases increase where it is difficult to reproduce the defect of applications. In this study, we propose (1) the platform realizing data exchange between applications via real-time database (RTDB) and (2) the method of reproducing the timing of data input with regard to execution of applications. We evaluated the proposed method and the result showed the method has probability of reducing the time taken to reproduce the defect applications by about 60% compared with the conventional method.

Keywords: automated driving, ECU, development platform

1. はじめに

自動運転等の走行制御システムの急速な進展にともない，従来に増してテスト効率化の重要性が高まっている．走行制御システムでは，あらゆる外部環境において正しく動作することを検証する必要があるため，膨大な量の実環境走行やシミュレーションによるECU（Electronic Control

[1] 株式会社日立製作所研究開発グループ
Research & Development Group, Hitachi Ltd., Yokohama, Kanagawa 244–0817, Japan
[2] 日立オートモティブシステムズ株式会社技術開発本部
Technology Development Division, Hitachi Automotive Systems Ltd., Hitachinaka, Ibaraki 312–8503, Japan
[a)] yuki.horita.vr@hitachi.com

Unit）のテストが必要となる [1].

ECU のテスト効率化のためには，テスト実行の効率化と不具合解消の効率化が必要である．前者は，xILS（X In the Loop Simulation）等のシミュレーション技術 [2], [3] やテスト自動化技術 [4], [5], [6] の組合せにより，効率化が進められている．一方，後者では，開発者による不具合解析の容易化のため，ECU 内の演算処理の振舞いの再現が鍵となる．その手段としては，従来から ECU に対する外部入力データをログとして保存しておき，そのログを再生入力するという方法がとられてきた [7].

しかし，近年の走行制御システムの高度化にともない，ECU が取り扱うセンサや搭載する演算機能（以降，アプリケーション）の数が急速に増加しており，これらのアプリケーション・ソフトウェアを実行するマイクロコントローラ（マイコン）もマルチコア化が進んでいる．このため ECU の外部入力を再現したとしても，ECU 内部のアプリケーションの振舞いの再現が難しいケースが増えてきている．これは，ECU の外部入力が同一でも，それに対するアプリケーションの演算処理のタイミングのずれや，アプリケーション間の相互作用により，ECU やマイコン内部の振舞いが変わることに起因する．外部入力やアプリケーション数が増加するほど，ECU 内部の振舞いを再現するのが難しくなる．再現性の低い不具合は解消に時間を要するため，ECU 内部のアプリケーションの振舞いの再現性を高める手段の確立が課題になっている．

再現性を高めるには ECU 内部のアプリケーションに対するデータ入力を制御する必要がある．しかしながら，従来の ECU 向けプラットフォームでは，アプリケーション間で個別にデータを授受する場合が多く，統一的な仕組みがないため，アプリケーションに対するデータ入力を制御するのが困難だった．ECU 外部でデータの再生タイミングを細かくずらして入力する仕組み [8] の構築により，タイミング起因の不具合をある程度洗い出すことは可能だが，やはり検出された不具合の再現を制御するのは難しい．

そこで，本稿では，ECU 上のアプリケーションの振舞いの再現を容易化するプラットフォームを提案する．提案するプラットフォームは，文献 [9], [10] で提案されたアーキテクチャの考え方をベースに，上記課題および ECU 環境向けに拡張したもので，以下の特徴を有する．

(1) リアルタイムデータベース（RTDB）によるアプリケーションのデータ入出力の統一的な制御
(2) アプリケーション実行を基準としたデータ入力タイミングを高精度に再現するログ記録・再生制御

以降では，以下の構成に従って説明する．2 章で関連研究を述べる．3 章および 4 章で提案する ECU プラットフォームおよびログ記録・再生制御方式の詳細を説明する．5 章で提案方式を実装して評価した結果を述べ，最後に 6 章でまとめる．

2. 関連研究

ROS（Robot OS）[11] は，ロボットの制御アプリケーション開発を目的としたオープンソースフレームワークで，自動運転向け開発用プラットフォームとして広く利用されている．現在の自動運転開発ブームの火付け役ともいえる DARPA Urban Challenge（2007 年）で自動運転システム開発向けに利用され，その実績以降，数多くの自動運転向け開発プラットフォームとして採用されている [12].

ROS では，アプリケーション間のデータ受渡しは，トピックと呼ばれるメッセージを介して行われる．トピックは Publisher/Subscriber メッセージングモデルに基づいて送受信されるため，各アプリケーションは通信先を意識する必要がなく，柔軟なアプリケーションの追加・削除が可能である．また，周辺の認識情報を可視化するツール（RViz）や，トピックの記録や再生を可能とするツール（rosbag）等が整備されており，アプリケーションのデバッグも柔軟に行える [11].

他方で，Publisher/Subscriber メッセージングモデルと同様な思想を，リアルタイムデータベース（RTDB）を用いて実現するアーキテクチャが文献 [9], [10] で提案されている．本アーキテクチャでは，KogMo-RTDB と呼ばれる RTDB がアプリケーション間のデータ受渡しを仲介する．それにより，ROS と同じようにアプリケーションの追加・削除，入出力データの記録・再生を柔軟に行うことが可能である．メモリアクセスによりデータ受渡しを実現するため，ROS と比較してリアルタイム性を確保しやすいという利点がある．また，文献 [13], [14] では，KogMo-RTDB をベースに，Publisher/Subscriber 型の通信ミドルウェア IceStorm と連携することで，複数ロボットの連携システムの開発を可能とするフレームワーク（ARCADE）も提案されている．

これらのプラットフォームは効率的な開発・デバッグを可能とし，研究や機能検証を目的とした開発フェーズで効力を発揮する．一方で，製品化を目的とした ECU に対しては，リソース制約や安全要求等を考慮して設計する必要があり，適用されていないのが現状である．ROS は，リアルタイム性保証やリソース制約のある環境での動作保証が困難である [15]．従来の RTDB は通常任意のデータを格納できるように動的にメモリ管理する機構を備えるが，ECU では安全性の要求から禁止されている．静的にメモリ配置を定めるようにするためには，事前に RTDB 内でデータを格納するメモリ構造を RTDB 内部で記述しておく必要があり，アプリケーションの修正・追加のたびに様々な箇所でコード修正が発生し，保守性が下がるという課題がある．

一方，代表的な ECU 向けプラットフォームとしては，AUTOSAR [16] があげられる．AUTOSAR では，ECU のソフトウェアのアーキテクチャや基本機能モジュール（BSW）の仕様を標準化し，異なるベンダ間の相互接続を

可能とすることにより，業界内のソフトウェアの再利用性を高めている．また，BSW やアプリケーションモジュール（SW-C）の接続形態を規定する設計情報から，実行プログラムのコード（RTE）を自動生成する開発プロセスも規定している．そのため，静的設計の制約の中でも，高い保守性を実現することを可能としている．

しかし，すべてのアプリケーション間のインタフェースを 1 対 1 で規定する必要があるため，アプリケーションの入出力の可視化や記録・再生を，ROS や KogMo-RTDB のように統一的に制御するのが困難である．そのため，ECU に対する外部入力データのログを再生してアプリケーションの不具合を再現する従来のデバッグ方式が主流である．

以上のように，既存のプラットフォームでは，ECU 上のアプリケーションのデバッグを支援する環境を体系的に提供できていない．

3. 提案プラットフォーム

提案するプラットフォームのアーキテクチャを図 1 に示す．本アーキテクチャでは，KogMo-RTDB [9], [10] のアーキテクチャと同様，リアルタイムデータベース（RTDB）がアプリケーションの入出力データを一元管理する構成になっている．ただし，ECU 環境への適用のため，RTDB は静的設計を前提とし，設計情報に基づいて設定情報やプログラムコードをツール（Customization Tool）により自動生成する仕組みを備える．また，開発用 PC の実行基盤である ROS と接続する機能（ROS 接続制御：ROS Linker）と，そのデータ入出力を制御して ROS ツールとシームレスに連携可能とする機能（データ入出力制御：Data I/O Controller）を備え，アプリケーションのデバッグを効率化する環境を提供する．

3.1 リアルタイムデータベース（RTDB）

本アーキテクチャでは，アプリケーションは RTDB を介してデータの入出力を行うため，RTDB のデータ処理速度が，ECU のリアルタイム性を保証するうえで課題となる．そのため，RTDB にはデータ処理が軽量な KVS（Key Value Store）方式を採用しており，μ 秒オーダのデータ処理が可能となっている [17]．RTDB の API は，ID 指定の検索・登録・削除のほか，格納データのフィールドを用いた条件式による検索をサポートしており，アプリケーションは演算に必要なデータを RTDB から柔軟に取得可能である．

また，本プラットフォームは，アプリケーションの入出力処理が RTDB の API で統一化されるため，以下の 3 つの特性を備える．

(1) 任意のアプリケーションのデータ入出力に対する共通処理の組込み，制御が可能．
(2) アプリケーション間のインタフェースは，RTDB が管理するデータ仕様定義により規定可能（従来は，受渡し対象のデータ仕様だけでなく，個別に関数仕様の定義も必要）．
(3) 各アプリケーションはデータの入出力先を意識する必要がない．

これらの特性により，従来の ECU 向けプラットフォームでは困難だった，アプリケーションの入出力データの可視化やログ記録・再生等のデバッグ機能に関するフレームワーク化（3.3 節）が可能となる．

本プラットフォームの RTDB の特徴は，静的設計に対応している点である．AUTOSAR の思想と同様，設計時に構築される RTDB で管理するデータ仕様（データ構造，格納数等）の定義情報に基づいて，RTDB 内部の該データ仕様に依存するプログラムコードが自動生成される．そのため，動的メモリ割当てが禁止されている ECU 環境にも適用可能である．

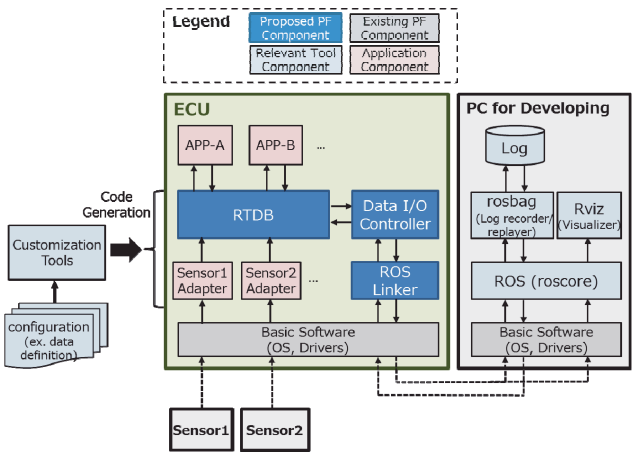

図 1　提案プラットフォームアーキテクチャ
Fig. 1　Proposed platform architecture.

3.2 ROS 接続制御（ROS Linker）

ソフトウェアのテスト用ツール群は，開発プロセスにおいて共通化されていることが望ましい．たとえば，ソフトウェアの機能設計のフェーズで用いていたツール群が，ECU 搭載後に利用できない場合，再度同様のツール群を ECU 環境向けに作り直す必要がある．また，ECU で発生した不具合を ECU 上で解析するのは限界があるため，開発環境が整っている機能設計環境で再現して解析できることが望ましいが，ツールが整合していないと難しい．

2 章で述べたように，ROS は研究や機能検証のフェーズで採用実績の多い有用な開発プラットフォームである．また，自動車業界におけるシステム設計・開発に用いられている MATLAB/Simulink [18] でも，ROS と連携する Robotics System Toolbox [19] がリリースされており，テスト用ツールとして ROS の RViz や rosbag 等が利用されるケースが今後増えていくと想定される．

そこで，本プラットフォームでは，ROS の Publisher/Subscriber 型の通信プロトコルを搭載し，他装置で動作する ROS のネットワークへの接続を可能とする．これにより，RViz や rosbag 等の ROS のテスト用ツールやアプリケーションとの連携が可能である．

3.3　データ入出力制御（Data I/O Controller）

ROS 接続制御は，ECU 外部の ROS 環境と通信可能とする機能であり，RViz による可視化や rosbag によるログ記録・再生を実現するためには，該当データを適切に入出力するプログラムが必要である．これらは本来アプリケーションに依存する部分だが，本プラットフォームでは，データ入出力制御が，RTDB と ROS 接続制御の間のデータ入出力を統一的に制御することで，アプリケーションに意識させずに，可視化やログ記録・再生の制御を実現する．

データ入出力制御は，以下の 4 つのサブ機能で構成される．

(a)　可視化データの出力

設定情報に基づき RTDB から可視化対象データを定期的に抽出し，開発用 PC 上の RViz に対してトピック送信する．文献 [17] に記載のように，リアルタイムデータベースに格納するデータ構造において，各データ（たとえば，センサデータ）に共通する項目（ID，種別，相対位置，相対速度等）を共通ヘッダとして持たせることにより，RTDB に格納されるデータの種類が追加・削除されても，可視化に必要な情報を自動抽出して RViz に出力することが可能である．

(b)　RTDB への入力操作ログデータの出力（ログ記録）

各アプリケーションにより RTDB への入力操作 API（登録・削除等）の呼び出し時に，該操作の引数情報や呼び出し元アプリケーションの ID，タイムスタンプ等を，ROS 接続制御を通じてトピック送信する．トピックとして出力されたデータは，開発用 PC 上の rosbag により操作ログデータとして保存される．これにより，各アプリケーションの出力データを自動記録することが可能である．

(c)　RTDB への入力操作ログデータの入力（ログ再生）

RTDB への入力操作ログデータは，rosbag により再生可能である．データ入出力制御機能は，ROS 接続制御を通じて rosbag の再生データを受信し，RTDB の操作の引数に復元して該当する API を実行する．

本プラットフォームでは，各アプリケーションは RTDB を介してデータの受渡しを実行するため，格納されているデータの生成元を意識することはない．そのため，再生されたログデータに基づき RTDB の格納データを再現すれば，特にテスト用のプログラムを個別に用意したりアプリケーションを修正したりする必要がなく，アプリケーションに対する入力データを再現できる．

ログ再生時には，あわせて再生対象のアプリケーションの停止が必要である．アプリケーションによる RTDB への入力操作とログによる入力操作が同時に発生するとデータが重複入力され，本来とは異なる挙動になる．本プラットフォームでは，後述のように，設定情報に基づいてアプリケーションの RTDB への操作の実行を ON/OFF 制御することにより，同種データの重複入力を回避する．

以上の方式により，プログラム修正なく設定情報の切替えだけで，ログ再生によるアプリケーション単位の演算処理の再現が可能である．これはプログラム修正が困難な ECU 環境に適用するうえで重要な特性である．

従来の ECU プラットフォームでは，ECU に対する外部入力データを再生して，内部処理をすべて実行することでアプリケーションの挙動を再現する．それに対し本プラットフォームでは，RTDB を介してアプリケーションに対する入力データを直接制御することができるため，より高精度にアプリケーションの挙動を再現できる．アプリケーションの挙動の再現率を高めるログ記録や再生の具体的方式については，4 章で述べる．

(d)　設定情報管理

以上に述べたサブ機能は，すべてのアプリケーションに対して適用すると，ECU の負荷が重くなる懸念がある．そのため，適用対象を柔軟に選択できることが望ましい．

データ入出力制御機能は，内部で設定情報を管理しており，その設定情報に基づいて各サブ機能の実行を制御する構成になっている．設定情報は，ROS の Parameter Server の機能を用いて，外部から変更可能である．

表 1 は通常実行時の設定情報の例を示したものである．設定情報は，アプリケーション単位で以下の項目を設定する構成である．

- **処理実行（Execution）**：該当アプリケーションの RTDB に対する入力操作を実行するか（Yes：実行，No：非実行）．
- **ログ出力（Log Record）**：該当アプリケーションの RTDB に対する入力操作をログ出力するか（Yes：出力，No：非出力）．
- **ログ入力（Log Replay）**：該当アプリケーションの RTDB に対する入力操作に関する再生ログを RTDB に入力するか（Yes：入力，No：非入力）．なお，処理実行とログ入力は排他関係になるため，処理実行の設

表 1　設定情報例（通常実行時）

Table 1　Example of configuration information (Normal execution).

APP ID	Execution	Log Record	Log Replay
Sensor 1 Adapter	Yes	Yes	-
Sensor 2 Adapter	Yes	Yes	-
APP-A	Yes	Yes	-
APP-B	Yes	No	-

表 2　設定情報例（APP-B デバッグ時）
Table 2　Example of configuration information (APP-B debugging).

APP ID	Execution	Log Record	Log Replay
Sensor 1 Adapter	No	No	No
Sensor 2 Adapter	No	No	Yes
APP-A	No	No	Yes
APP-B	Yes	Yes	-

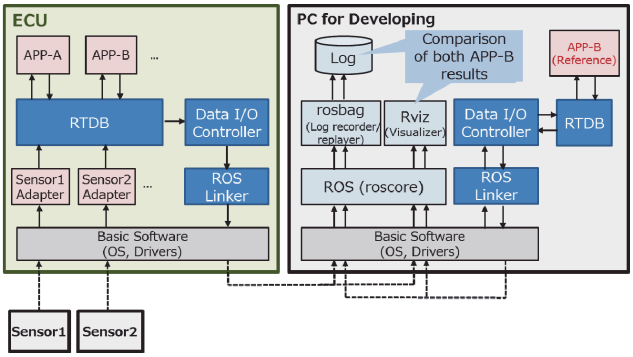

図 2　開発用 PC 上での APP-B 監視
Fig. 2　APP-B monitoring on development PC.

定が「Yes」の場合は，無条件で無効化（表中の「-」）される．

続いて，ログ再生により APP-B をデバッグする際の設定情報の例を**表 2** に示す．APP-B は，センサ 2 アダプタと APP-A の出力情報を用いて演算処理を行う想定であるが，APP-B のみ RTDB の処理実行を有効とし，その入力データであるセンサ 2 アダプタと APP-A のログ入力を有効にする設定にすることにより実現可能である．

この設定情報の切替えにより，ログ再生によるデバッグだけでなく，様々なテスト方式に応用することも可能である．たとえば，開発用 PC 上でも本プラットフォームを動作させてアプリケーションを実行することもできるので，**図 2** に示すように APP-B を開発用 PC 上で監視実行することも可能である．これは開発用 PC 上のプラットフォームが，ECU からのログ出力を再生データのように扱うことが可能なため，それを開発用 PC 上の APP-B に再生入力させて ECU 上と同じ挙動を外部で再現させることができる．それ以外にも同じ仕組みを活用して，ECU と開発用 PC の連携実行等，状況に応じた様々なテスト方式を柔軟に行うことができる．

4.　ログ記録・再生制御方式

本章では，提案プラットフォームにおけるデータ入出力制御で行う (b) ログ記録，(c) ログ再生の具体的方式について説明する．そのために，まず 4.1 節で従来のログ記録・再生制御方式の課題について述べ，その後で提案するログ記録・再生制御方式について述べる．

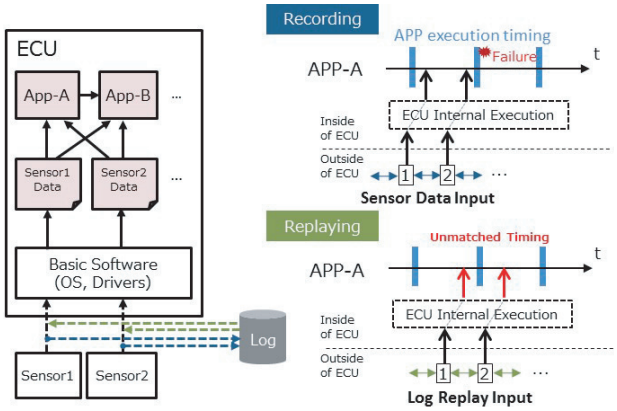

図 3　従来のログ記録・再生方式の課題
Fig. 3　Issue of conventional log record and replay.

4.1　ECU のバグ解析における課題

従来から ECU 内部のアプリケーションの振舞いを再現してバグ解析するために，ECU に対する外部入力データをログとして保存しておき，そのログを再生入力するというアプローチがとられてきた．しかし，近年の走行制御システムの高度化にともない，ECU が取り扱うセンサや搭載するアプリケーション数が急速に増加しており，ECU の外部入力を再現しても，ECU 内部のアプリケーションの振舞いの再現が難しいケースが増えてきている．

図 3 は，従来方式のログ記録・再生の構成とアプリケーションの振舞いを再現できない例を示したものである．従来方式では，ECU 外部からのデータ入力のタイミングでログを記録する．ECU に対して入力するデータの時間間隔が記録時と同じになるようにログを再生することにより，アプリケーションの振舞いを再現する．

図 3 の右上図は，アプリケーション A（APP-A）の記録時（Recording）の振舞いを表したものである．一般的に ECU のアプリケーションの処理は周期的に実行される．図中の青い線は APP-A の実行タイミングを表現しており，ある処理実行において不具合が発生したことを示している．不具合を起こした処理の前では，センサ 1 とセンサ 2 のデータが APP-A に入力されている（図中の Sensor Data Input の 1 と 2）．

ここで，APP-A の不具合が直前に入力されたセンサ 1 とセンサ 2 のデータの組合せにより引き起こされたものとする．その場合，不具合を再現するためには，それらのデータを揃えて入力して APP-A を実行する必要がある．しかし，従来方式のログ再生では ECU に対して入力するデータの時間間隔しか再現できないため，APP-A の実行タイミングとログ再生のタイミングの関係によっては，図 3 の右下のログ再生時（Replaying）の図が示すように，APP-A 実行の際に対象のデータセットが揃わず不具合が発生しないケースが発生する．この例では，比較的単純なタイミングによる不具合モデルで示したが，実際にはより多くの

データの組合せにより発生する不具合も存在する．これらは発生頻度としては低いが，再現させるのがより困難となるため，バグ解析が難しく，またバグを解消したことを検証することも難しい．そのため，テスト期間を長期化させる主要因となりうるため，アプリケーションの不具合の再現性を高めるログ記録・再生手段の確立が課題である．

4.2　提案方式の概要

アプリケーションの振舞いの再現性を高めるためには，その処理に使われるデータ入力を再現することが求められる．しかし，従来方式では，ECU 外部のデータ入力を再現するため，ECU 内部のアプリケーションの実行タイミングに対して，記録時と同じデータを揃えて入力するように制御することは難しかった．一方，本プラットフォームでは，3.3 節で述べたように，データ入出力制御により，アプリケーションのインタフェースである RTDB に対して入力操作を直接制御することが可能である．そこで提案方式では，ログ再生時におけるアプリケーションの実行タイミングに対する RTDB への入力操作タイミングが，記録時と同じになるように制御することにより，上記課題を解決する．なお，本研究では，周期的に所定の演算処理を実行するアプリケーションを対象とし，演算処理開始時に当該処理に必要なデータが入力（取得）されるものとする．

提案方式では，ログ記録時に各データに対して下記 2 点の情報を関連付けて保存しておき，ログ再生時にそれらを再現するように RTDB に対する入力操作を制御する．

(1) 該入力操作時のアプリケーション実行サイクルの識別番号（サイクル番号）
(2) 直前のアプリケーション実行サイクルの開始時刻から該入力操作時までの経過時間（サイクルオフセット）

ここで「アプリケーション周期」という用語と「アプリケーション実行サイクル」という用語を定義する．前者は，特定のアプリケーションにおける実行の時間間隔（10 ms 等）を表すものとする．それに対し，後者は，すべてのアプリケーションに着目したときの，共通したアプリケーションの実行パターンの繰返しの時間間隔を表し，各アプリケーション周期の最小公倍数に該当する．たとえば，周期がそれぞれ 20 ms と 50 ms の 2 つのアプリケーションを実行する場合のアプリケーション実行サイクルは 100 ms である．なお，アプリケーションが 1 つの場合は，アプリケーション実行サイクルはアプリケーション周期と同等である．

本プラットフォームにおけるデータ入出力制御によるログ記録・再生の処理の流れを説明する．以下では，簡単のため単一アプリケーションを対象とした例で説明する．

図 4 はログ記録時のタイムチャートを示している．この図では，下から上に向かってセンサからの入力データが APP-A（実際は RTDB）に入力されるまでの流れを，左から右に向かって時間の流れを示している．ECU の外部から各センサの出力データが入力されると，受信処理やアダプタ処理等の内部処理を経て，RTDB に該データが入力される．その際，上述のようにデータ入出力制御により，RTDB に対する入力操作情報が ROS 接続制御を通じてトピック送信されるが，その中にあわせてサイクル番号とサイクルオフセットの情報を含めておく．サイクル番号は，たとえば，ECU 内で管理されるアプリケーション実行サイクルのシーケンス番号（図の①…）を取得することにより求められる．また，サイクルオフセットは，ECU 内で管理されるアプリケーション実行サイクルの開始時刻と該データの RTDB への入力時刻との差分により求められる．

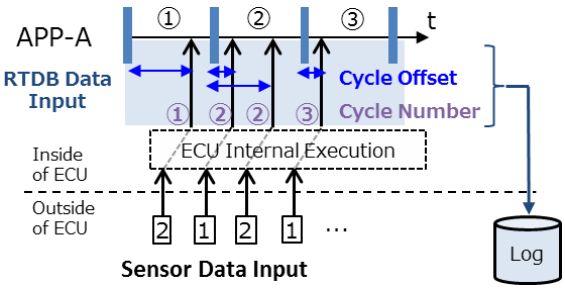

図 4　提案方式（ログ記録時）
Fig. 4　Proposed method.

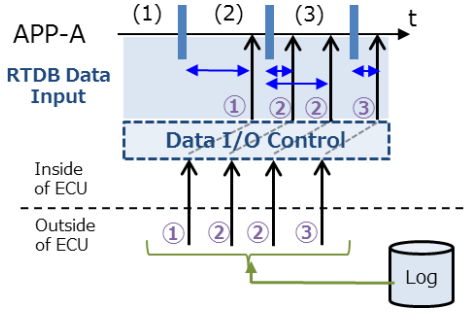

図 5　提案方式（ログ再生時）
Fig. 5　Proposed method.

図 5 はログ再生時のタイムチャートを示している．この図では，下から上に向かって，ECU 外部で再生されたログが APP-A（実際は RTDB）に入力されるまでの流れを，左から右に向かう時間の流れと合わせて表現している．データ入出力制御は，ECU 外部から再生された各入力操作情報を取得すると，その中に含まれるサイクル番号とサイクルオフセットを利用して，アプリケーションの実行サイクルを基準とした入力操作の実行タイミングを算出する．図 5 では，最初に到着したサイクル番号①の入力操作は APP-A の実行サイクル (2) に実行されている．この「入力操作のサイクル番号 + 1 = APP-A の実行サイクル」という関係を，以降のログにも適用することにより，ログ記録時に各アプリケーション実行サイクルの間に実行された入力操作を再現可能である．また，該当するアプリケーション実行サイクルの開始時刻からサイクルオフセット分

の時間を待ってから入力操作を実行し，ログ記録時のサイクルオフセットも再現する．これらにより，再生時のアプリケーション実行タイミングに対する，RTDB への入力操作のタイミングを，記録時と同等となるよう制御でき，各アプリケーションの入力データを再現できる．

なお，厳密には，RTDB に対する入力操作を再現しても，RTDB の内部状態を復元できないと，各アプリケーションに対する入力データを再現できない場合がある．今回対象としている走行制御向けアプリケーションの場合，センサから継続して出力される時系列データの中で，最新値と（多くても）その過去数回分の値が処理対象となり，それ以上古いデータは使われない．そのため，本プラットフォームの RTDB は，最新の所定回数分のセンサデータしか保存しない仕組みになっており，不具合の事象発生時点よりも十分に前もって入力操作を再現していけば，不具合発生時の RTDB の内部状態を復元可能である．それでも仮に RTDB 内に更新頻度が著しく低い状態値があり，直近のログ再生だけでは状態の復元が困難な場合は，チェックポイントとして定期的に該当する状態値を取得，ログ出力する機構を設けるという方式も考えられる．この方式を適用することで，所定時間のログ再生で RTDB の内部状態を復元することを保証することが可能となる．

上記では単一のアプリケーションを対象に説明したが，本方式を複数のアプリケーションが実行される環境への適用も可能である．アプリケーションが単一の場合と複数の場合で異なるのは，各アプリケーションが，ECU 外部から入力されるセンサデータだけではなく，他アプリケーションの演算結果も用いて，演算処理を行う可能性があるという点である．3 章の提案プラットフォームでは，アプリケーションの入出力は RTDB を介するため，他アプリケーションの演算結果（出力）もログに記録されている．そのため，上述した単一アプリケーションのモデルをそのまま適用可能で，他アプリケーションの演算処理結果も含めた入力データ再現により，任意のアプリケーションの振舞いを再現することが可能である（3 章の表 2 が具体例）．

5. ログ記録・再生のタイミング制御方式の評価

3 章の提案プラットフォームに従い 4 章の提案方式を実装し，以下 2 点について従来方式との比較評価を行った．
(1) アプリケーションに対するデータ入力の再現率（実験 1）
(2) 不具合再現に必要な試行回数（実験 2）

4.1 節で述べたアプリケーションに対する複数のデータ入力操作の組合せを揃えるという課題に対して，実験 1 では提案方式がどの程度の精度で実現できるかを，実験 2 ではそれによる不具合解析が効率化される度合いを，それぞれ評価することを目的とする．

なお，4.2 節で述べたように，複数アプリケーションが動作する環境においても，所定アプリケーションの不具合解析は，単一アプリケーションのモデルに帰着されるため，いずれの実験でも単一アプリケーションを用いて評価した．

表 3　評価環境
Table 3　Experimental environment.

Hardware	CPU	Intel Core i5-4310U 2.0GHz 32Bit
	RAM	4 GB
Software	OS	Linux Ubuntu 14.04
	ROS	Indigo
	Compiler	gcc 4.8.4 (C Language)

5.1　評価環境

表 3 に本評価実験において用いたハードウェアおよびソフトウェアの環境を示す．

提案プラットフォームは，本来はリアルタイム OS を搭載した ECU 上で動作する．それに対し，今回の評価環境では Linux を用いているため，タスクのスケジューリング遅延の影響等により厳密な実行制御が難しく，実際の ECU 環境とは異なる挙動を示す可能性がある．

一方，今回の評価実験の目的は，従来方式に対する提案方式の再現率に関する比較評価である．4 章で述べたように，従来方式と提案方式における再現率の差異は，アプリケーションの実行タイミングに対する再生データの入力タイミングのズレ（ms 単位）に起因するものである．このズレによる再現率に対する影響と比較すると，スケジューリング遅延（≪ ms）による影響は軽微である．そのため，本実験環境における評価結果は，ECU 環境においても従来方式に対する提案方式の効果として十分な示唆を与えるものと考えられる．

5.2　実験 1：データ入力の再現率の評価

5.2.1　実験概要

本実験では，サイクルあたりの入力操作数をパラメータとして変化させた際の，アプリケーションの実行サイクルごとの入力操作の再現率を，提案方式と従来方式で比較評価する．

本実験で評価する再現率の定義は，「ログ記録時とログ再生時において，あるアプリケーション実行サイクルにおける入力操作が同一となる確率」とする．たとえば，着目したサイクルに入っていた入力操作が 3 つあった場合には，ログ再生時に，それらの入力操作のみが同一サイクルに行われることを再現の条件とする．すなわち，当該入力操作が複数のサイクルに分かれたり，記録時に異なるサイクルで行われた別の入力操作が，当該入力操作と同じサイクルに混在したりした場合は，再現できなかったものとする．

走行制御システムでは，アクチュエータ制御等のような高頻度に（たとえば 10 ms）処理する必要があるものから，センサ情報から周辺環境を認知するセンサフュージョン等

のように比較的低頻度（たとえば 50 ms）に処理するものまで，多様な実行周期のアプリケーションが存在する．今回はその中で最もタイミング制御の精度が求められる高頻度実行のアプリケーションを想定し，実行周期を 10 ms と設定した．

また，走行制御システムは，今後の自動運転の高度化にともない，車両に 10 個以上のカメラやレーダ等のセンサが装着されることが想定されており，各センサからは複数種類の認識情報（車両，歩行者，白線，信号等）が数 10 ms〜100 ms 周期で出力される．それらのセンサ情報に加え，車内センサ（速度，加速度等）の情報も，同様の周期で ECU に入力されるため，たとえばセンサフュージョンのようなアプリケーションでは，多い場合で 1 サイクル中に 100 近くの関連する入力操作が発生することがありうる．そこで，サイクルあたりの入力操作数の範囲は 1〜100（実測点：1，3，5，10，50，100）とした．

入力操作数分の各入力操作に対してサイクルオフセットを 0 ms 以上 10 ms 未満の範囲でランダムに生成し，それぞれ 1 万回ずつ試行した場合の再現率を求めた．比較対象の従来方式は，本プラットフォームにおいて，データ入出力制御がタイミングの制御を行わず，ログを受信すると同時に RTDB 入力操作を行うこととした．

5.2.2　結果と考察

従来方式および提案方式の再現率の結果を**図 6** に示す．従来方式では，入力操作数が 1 の 0.675 をピークに，入力操作数が大きくなると急激に再現率が下がり，やがて 0 に漸近していく．従来方式では，アプリケーションの実行タイミングに対して，再生データの入力タイミングがランダムにずれるため，対象の入力操作が異なるサイクルに分かれたり，本来別サイクルで行われるべき入力操作が当該サイクルに混在したりする．この事象は，関連する入力操作がサイクルの切替りタイミングの近くに存在するほど，上記入力タイミングのズレによる影響を受けやすく，発生確率が高まる．そして，入力操作数が増えるほど，関連する入力操作がサイクルの切替りタイミング付近に存在する確率が上がるため，入力操作数が増えると急激に再現率が下がっていくと考えられる．

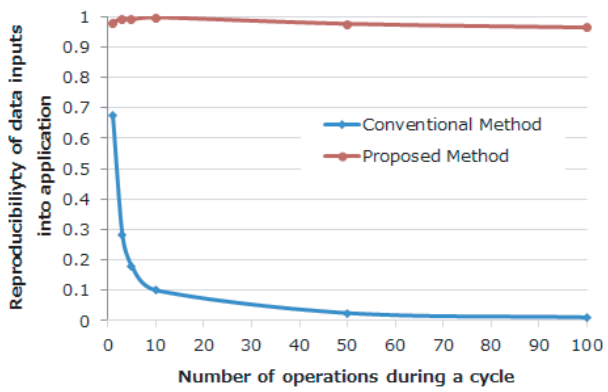

図 6　提案方式と従来方式の再現率

Fig. 6　The reproducibility of conventional method and proposed method.

上述のように走行制御向けアプリケーションでは，多数のセンサデータの組合せを処理することになり，今後そのデータ数は増加傾向にある．これをふまえると，従来方式では，記録時のアプリケーションの演算処理を再現するのは今後難しくなり，不具合の再現が難しいケースも増えていくと考えられる．

それに対し，提案方式では，入力操作数にかかわらず，高い再現率（96%以上）を維持できている．アプリケーションの実行タイミングに合わせて再生データを入力するため，入力操作数の影響を受けずに，ほぼ確実に指定のサイクル内に対象の入力操作を行うことができるためである．

提案方式においても，入力操作数が増えると再現率が 100%から徐々に減少しているが，これは再生時における入力操作を行うサイクルオフセットの制御誤差による影響と考えられる．指定したサイクルオフセットとログ再生時の実際のサイクルオフセットのズレを評価したところ，$6.2 \pm 138.4\,\mu$s のズレが生じることが分かった．このサイクルオフセットのばらつきは，Linux 上の別プロセスによる割込みやタイマ誤差による影響で発生しているものと考えられる．それに対しリアルタイム OS 上で動作する ECU 環境では，制御誤差はさらに小さく抑えられると考えられため，再現率は上記結果よりも改善されると推察する．

5.3　実験 2：不具合再現に必要な試行回数の評価

5.3.1　実験条件

実験 1 により，提案方式によりアプリケーションに対するデータ入力の再現率が向上することを示した．本実験では，それにより不具合再現にかかる時間をどの程度削減できるかを評価するため，想定されるアプリケーションの不具合モデルにおいて，不具合を再現するまでの試行回数を計測する．

本実験では，不具合モデルとして，アプリケーションに入力された特定の N 個（$N \geq 2$）の入力操作の組合せにより引き起こされる不具合を想定する．該当するすべての入力操作が同一サイクルに含まれている場合は不具合が再現し，そうでない場合は不具合が再現しないものとする．本実験では，実環境において実際に発生する可能性が高いと考えられる $N = 2$〜4 のケースについて評価した．

10 ms で動作する単一のアプリケーションを想定し，0 ms 以上 10 ms 未満の範囲でサイクルオフセットをランダムに生成した N 個の入力操作を 100 セット用意し，従来方式および提案方式において，すべてのセットにおいて不具合が再現するまでの総試行回数を計測，比較した．これは，開発工程で発生した上記モデルに従った不具合 100 件を，どれぐらいの延べ時間で再現できたかの評価に相当する．

5.3.2　結果と考察

従来方式および提案方式において，100 個の不具合が再現するまでの総試行回数を**表 4** に示す．

従来方式では，$N = 2$ の場合で 242 回要し，N が増えるとさらに総試行回数が増加している．それに対し，提案方式ではすべての不具合を 1 度で再現することができた．これは従来不具合再現にかけていた時間を，提案方式により少なくとも 58.7%（$N = 2$ の場合）削減できる試算となる．

続いて，従来方式における入力操作の各セットに対して再現するのにかかった試行回数のヒストグラムおよび累積値をそれぞれ**図 7**，**図 8** に示す．

これらのグラフから，大半の不具合は少ない試行回数で再現していることが分かる．たとえば，試行回数 5 回以内で再現可能な不具合は，$N = 2$ の場合で 95%，$N = 3$, 4 の場合でも 70%を超えている．つまり，ほとんどの不具合は従来方式でも大きな問題なく対応可能であるといえる．一方，再現が難しい不具合（試行回数を 10 回以上）を見ると，個数は全体の中のわずか（$N = 2$：5%，$N = 3$：12%，$N = 4$：13%）にすぎないが，総試行回数では約半分（$N = 2$：39%，$N = 3$：55%，$N = 4$：54%）を占めた．このように，再現するのが困難な不具合が少しでも含まれていると，テスト効率に大きな影響を与える．提案方式では，再現が難しい不具合に対しても 1 回の試行で再現可能のため，不具合解析に要する時間を大きく削減することができると考えられる．

なお，N を増やした場合の各不具合に対する試行回数の分布に着目すると，$N = 2$ に対して $N = 3$ では試行回数が多くなる方向に分布が大きく変化しているのに対し，$N = 3$ と $N = 4$ ではほぼ同等に見える．従来方式では，不具合に関連する入力操作がサイクルの切替りタイミングの近くに存在するほどデータ再生時の再現は難しくなる．実験 1 の結果を見ると，入力操作数 N が増加すると急激に再現率が下がるが，$N = 3$ 以降は再現率の下降度合が緩やかになっている．そのため，$N = 3$ と $N = 4$ では試行回数の分布に大きな差異が見られなかったものと考えられる．

表 4　不具合再現に要した総試行回数
Table 4　Total number of iterations for reproducing defects.

	Total number of iterations for reproducing 100 defect samples		
	N=2	N=3	N=4
Conventional method	242	545	649
Proposed method	100	100	100

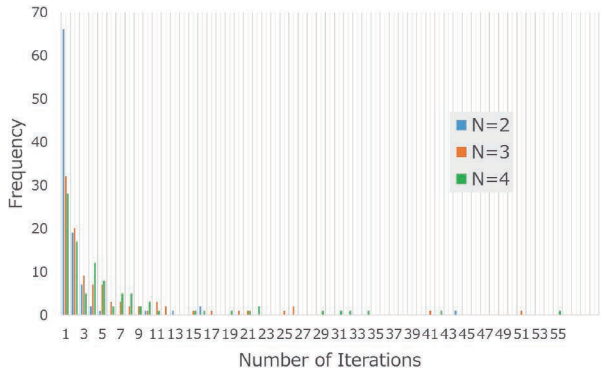

図 7　従来方式における各不具合の再現試行回数
Fig. 7　Number of iterations for reproducing each defect when using conventional method.

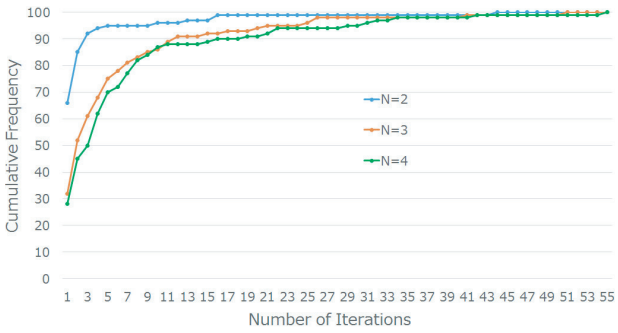

図 8　従来方式における各不具合の再現試行回数（累積）
Fig. 8　Number of iterations for reproducing each defect when using conventional method (cumulative).

6.　まとめ

本稿では，自動運転等高度な走行制御システムに向けて，アプリケーションの振舞いの解析や再現をしやすい ECU 向け開発プラットフォームを提案した．本プラットフォームは以下の特徴を持つものである．

(1) リアルタイムデータベース（RTDB）によるアプリケーションのデータ入出力の統一的な制御
(2) アプリケーション実行を基準としたデータ入力タイミングを高精度に再現するログ記録・再生制御

また，(2) の提案方式の評価を行い，従来方式と比べてアプリケーションの不具合再現にかかる時間を 58.7%削減できる見込みを得た．

今後は，提案したプラットフォームの実 ECU 上での評価，実車での評価を進めていく予定である．

参考文献

[1] Berger, C. and Rumpe, B.: Autonomous Driving – 5 Years after the Urban Challenge: The Anticipatory Vehicle as a Cyber-Physical System, *Proc. INFORMATIK 2012*, pp.789–798 (2012).
[2] Zofka, M.R., Klemm, S., Kuhnt, F., et al.: Testing and validating high level components for automated driving: Simulation framework for traffic scenarios, *Proc. IEEE Intelligent Vehicles Symposium* (*IV2016*), pp.144–150 (2016).
[3] Zhao, D., Liu, Y., Zhang, C., et al.: Autonomous driving simulation for unmanned vehicles, *2015 IEEE Winter Conference on Applications of Computer Vision* (*WACV*), pp.185–190, IEEE (2015).

[4] 有馬仁志，清水圭介：モデルベース開発ツールとプラントモデル ASM，計測と制御，Vol.53, No.4, pp.346–351 (2014).
[5] Berger, C. and Rumpe, B.: Engineering autonomous driving software, *Experience from the DARPA Urban Challenge*, pp.243–271 (2012).
[6] 藤原貴之，岡本周之：大規模組込み機器向けテスト自動化拡大方式，情報処理学会論文誌：コンシューマ・デバイス&システム，Vol.4, No.4, pp.1–10 (2014).
[7] Pallierer, R., Horauer, M., Zauner, M., et al.: A generic tool for systematic tests in embedded automotive communication systems, *Proc. Embedded World Conference 2005* (2005).
[8] 金澤　明，古戸　健，松本達治：車載ソフトの広範囲な起動タイミングの検証，SEI テクニカルレビュー，Vol.172, pp.106–111 (2008).
[9] Goebl, M. and Färber, G.: A Real-Time-capable Hard- and Software Architecture for Joint Image and Knowledge Processing in Cognitive Automobiles, *Proc. IEEE Intelligent Vehicles Symposium* (*IV2007*), pp.734–740 (2007).
[10] Goebl, M. and Färber, G.: Interfaces for Integrating Cognitive Functions into Intelligent Vehicles, *Proc. IEEE Intelligent Vehicles Symposium* (*IV2008*), pp.1093–1100 (2008).
[11] ROS.org, available from ⟨http://www.ros.org⟩.
[12] Kato, S., Takeuchi, E., Ishiguro, Y., et al.: An open approach to autonomous vehicles, *IEEE Micro*, Vol.35, No.6, pp.60–68 (2015).
[13] Althoff, D., Kourakos, O., Lawitzky, M., et al.: An Architecture for Real-Time Control in Multi-Robot Systems, *3rd International Workshop on Human Centered Robot Systems*, pp.43–52 (2009).
[14] Rambow, M., Rohrmüller, F., Kourakos, O., et al.: A Framework for Information Distribution, Task Execution and Decision Making in Multi-Robot Systems, *IEICE Trans. Information and Systems*, Vol.93, No.6, pp.1352–1360 (2010).
[15] Berger, C. and Dukaczewski, M.: Comparison of Architectural Design Decisions for Resource-Constrained Self-Driving Cars – A Multiple Case-Study, *GI-Jahrestagung*, pp.2157–2168 (2014).
[16] AUTOSAR, available from ⟨https://www.autosar.org/⟩.
[17] 福元和真，福島悠史，林　正人ほか：自動運転 ECU への外界認識データ管理一元化に向けたデータベース適用検討と性能評価，自動車技術会 2017 年春季大会学術講演会講演予稿集，No.5-17, pp.144–149 (2017).
[18] MATLAB/Simulink, available from ⟨http://www.mathworks.com/⟩.
[19] Robotics System Toolbox, available from ⟨http://www.mathworks.com/products/robotics.html⟩.

堀田 勇樹

2006 年東京大学大学院情報理工学系研究科電子情報学専攻修士課程修了．同年（株）日立製作所システム開発研究所入社．C-ITS システム，自動運転プラットフォームの研究開発に従事．

宇治土公 雄介

2016 年東京大学大学院学際情報学府修士課程修了．同年（株）日立製作所入社．自動運転プラットフォームの研究開発に従事．

福島 悠史

2002 年武蔵工業大学大学院工学研究科修士課程修了．同年（株）日立ソリューションズ（旧，日立ソフトウェアエンジニアリング（株））入社．2013 年より車載 ECU ソフトウェア開発に従事．

成沢 文雄　（正会員）

1994 年東北大学工学部情報工学科卒業．1996 年同大学大学院情報科学研究科情報基礎科学専攻修士課程修了．同年（株）日立製作所日立研究所入社．2016 年より日立オートモティブシステムズ（株）．自動運転システムアーキテクチャ開発・検証技術の研究開発に従事．ACM，自動車技術会各会員．

林 正人

1987 年（株）日立製作所システム開発研究所入社，ルネサスエレクトロニクス（株）を経て，2016 年日立オートモティブシステムズ（株）に勤務．博士（情報科学）．モバイルネットワーク，衛星通信システム，アドホックネットワーク，C-ITS システム，無線通信システムの研究開発および車載ソフトウェアプラットフォームの開発に従事．IEEE，電子情報通信学会各会員．